Санкт-Петербургский государственный университет

Курлов Дмитрий Николаевич

Выпускная квалификационная работа

Моделирование движение манипулятора и управление, основанное на нейронных сетях

Уровень образования: бакалавриат

Направление 01.03.03 «Механика и математическое моделирование»

Основная образовательная программа СВ.5008.2017 «Механика и математическое моделирование»

Профиль «Биомеханика и робототехника»

Научный руководитель:

Доцент, Кафедра теоретической и прикладной механики

к. ф.-м. н., доцент А. С. Ковачев

Рецензент:

доцент В. Г. Быков

Санкт-Петербург

2021

Saint Petersburg State University
Mechanics and Mathematical Modelling
Biomechanics and Robotics

Kurlov Dmitry Nikolaevich

Graduation Project

Modelling manipulator motion and its control based on neural networks

Scientific Supervisor:
Associate Professor, Department of
Theoretical and Applied Mechanics
A. S. Kovachev

Reviewer:
Associate Professor V. G. Bykov

Saint Petersburg
2021

Оглавление

Введение

В современном мире сфера применения роботов манипуляторов расширяется. Их можно встретить не только на различных линиях производства, но и при работе на опасных объектах или в опасных для человека местах. В связи с этим, разрботка и модификация автоматического управления для робототехнических систем, к которым также относятся манипуляторы, актуальна на сегодняшний день.

Работа с робототехническими манипуляторами начинается с описания геометрии. Для описания геометрии будем использовать параметры Денавита-Хартенберга. [1]

В данной работе рассмотрен вопрос моделирования движения робота манипулятора. Моделирование движения разделяется на построение математической модели кинематики и динамики робота. Для моделирования кинематики строится последовательность матриц однородных преобразований, благодаря которой решается прямая задача кинематики для рабочей точки схвата. Задача динамики решается с помощью составления систем дифференциальных уравнений движения в форме Лагранжа-Эйлера [2].

При моделировании динамики предполагаются допущения, такие как идеальность сочленений в шарнирах, неподвижность первой вращающейся опоры, которая позваляет рассматривать 3-х звенный манипулятор, как 2-звенный, плоский манипулятор. В дальнейшем, от этого допущения можно будет отказаться

Решение таких систем аналитически довольно трудоёмко, а зачастую вовсе невозмно, поэтому моделирование было проведено численно [3]. Такой подход в данной работе позволяет решать более широкий класс задач. Например, численно можно без особо труда с заданной точностью решить систему нелинейных дифференциальных уравнений, которыми как раз описывается движение.

С математической точки зрения выбор оптимального управления - это решение задачи оптимизации [4]. Одним из классических способов решения задачи синтеза управления, является создание ПИД-регулятора и его настройка для конкретной системы [1]. Также в настоящее время в теории оптимизации широкое применение находят методы машинного обучения, в частности нейронные сети, которые также можно использовать в задачай синтеза управления.

Есть два основных подхода в применении алгоритмов машинного обученя в задачах построения управления: прямой подход и косвенный. Прямой подход заключается в

том, что сам регулятор представляет собой нейронные сеть. Косвенный подход состоит в использовании нейронных сетей для исправления шумов сигналов или для настройки параметров классических регуляторов, например, ПИД-регултора. В данной работе рассматривается второй ваариант применения нейроныых сетей.

Главная задача данной работы — выяснить, для каких задач робототехники можно применить нейронные сети. А также выяснить, являются они более эффективными стандартных методов или нет, и по каким аспектам.

Глава 1

Моделирование простейшего манипулятора

1.1. Постановка задачи и цели

Рассматривается плоский двузвенный робот манипулятор. Необходимо смоделировать его движение согласно физическим законам и построить управление, решающее задачу слежения за траекторией.

Основные цели: построить математическую модель движения и управления плоским двузвенным манипуялтором. Реализовать алгоритмы управления, использующие нейронные сети и классическое управление. Сравнить результаты.

1.2. Моделирование движения

1.2.1. Геометрия манипулятора

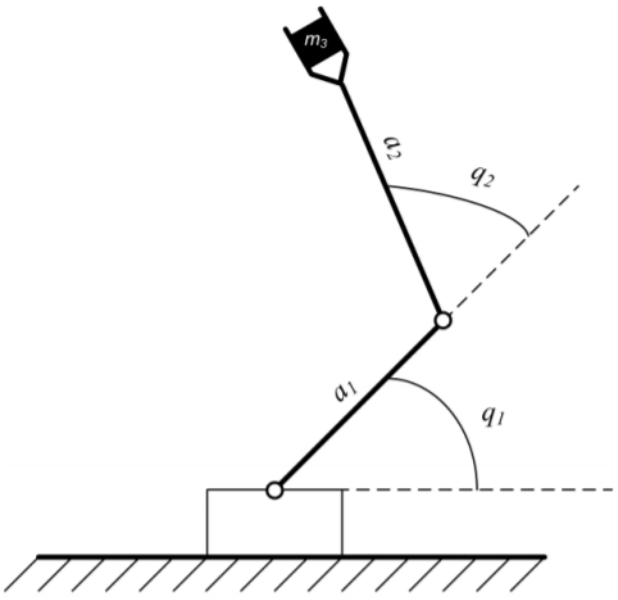

Рис. 1.1. Кинематическая схема двузвенного манипулятора

Рассмотрим двузвенный манипулятор, изображенный на рисунке 1.1, где за каждой кинематической парой, начиная от опоры, закреляется своя система координат $O_iX_iY_iZ_i$ по следующему правилу (см. рис. 1.2):

- Ось Z_i направлена вдоль оси $i+1$-го сочленения
- Ось X_i направлена вдоль общего перпендикуляра к осям Z_{i-1} и Z_i,
- Ось Y_i выбирается так, что система была правосторонней

Геометрическое представление n-звенного манипулятора можно построить с помощью 4-n параметров, которые называетются параметрами Денавита-Хартенберга [1]:

- d_i и θ_i расстояние и угол между общими перпендикулярами к осям Z_{i-1}, Z_i и Z_i, Z_{i+1}
- длина общего перпендикуляра к осям Z_i и Z_{i+1}
- α_i - угол между осями Z_i и Z_{i+1}

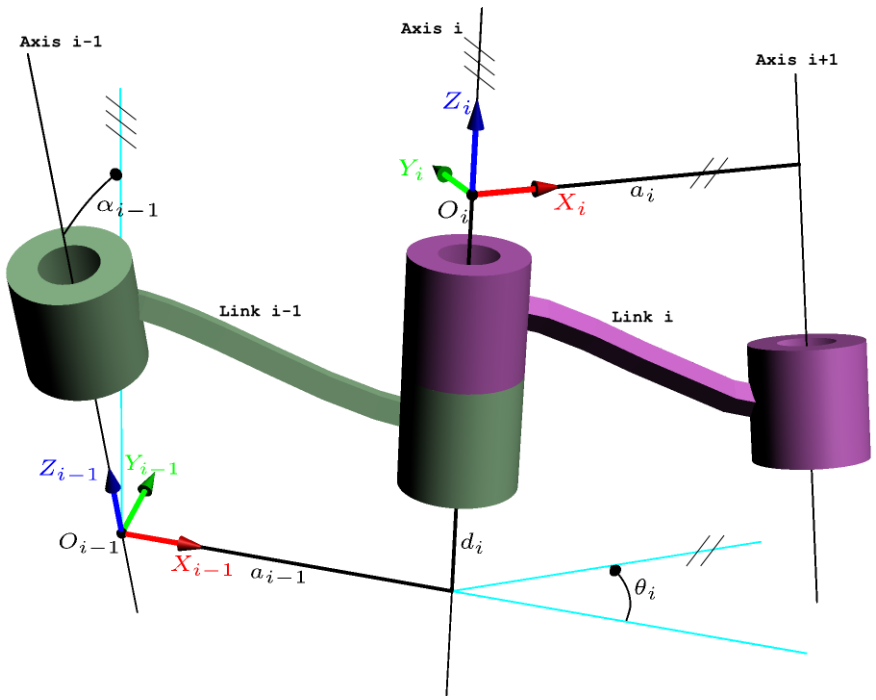

Рис. 1.2. Параметры Денавита-Хартенберга

Матрица параметров Денавита-Хартенберга для плоского двузвенного манипулятора представлена ниже. Элементы данной матрицы в общем случае завияст от обощенных координат.

	θ_i	d_i	a_i	α_i
1	q_1	0	a_1	0
2	q_2	0	a_2	0

Таблица 1.1. Параметры Денавита-Хартенберга для плоского двузвенного манипуялтора

1.2.2. Прямая задача кинематики

Для будущих расчетов нам необходимо уметь вычислять положения рабочей точки манипулятора. То есть, необходимо уметь решать прямую задачу кинематики - расчет декартовых координат рабочей точки по известным обощенным координатам. Будем делать это методом однородных преобразований [5]. Рассмотрим более общий случай для трехмерного манипулятора.

$$R_M^i = \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} = \begin{pmatrix} r_M^i \\ -- \\ 1 \end{pmatrix} \tag{1.1}$$

$$R_M^{i-1} = H^{i-1,i} R_M^i \tag{1.2}$$

Здесь R_M^i - расширенный вектор координат выбранной точки M в i-ой системе координат. Для

$$H^{i-1,i} = \begin{bmatrix} A^{i-1,i} & \vdots & r_i^{i-1} \\ \dots & \dots & \dots \\ 0 & \vdots & k \end{bmatrix} \tag{1.3}$$

Координаты схвата в системе координат, связанной с опорой, можно выразить через последовательность ортогональных преобразований:

$$R_M^0 = H^{0,1} H^{1,2} H^{2,3 i-1,i} R_M^i = H^{0,i} R_M^i \tag{1.4}$$

Таким образом, матрицу $H^{i-1,i}$ можно выразить следующим образом.

$$H^{i-1,i} = \begin{bmatrix} cos\theta_i & -cos\alpha_i sin\theta_i & sin\alpha_i sin\theta_i & a_i cos\theta_i \\ sin\theta_i & cos\alpha_i cos\theta_i & -sin\alpha_i cos\theta_i & a_i sin\theta_i \\ 0 & sin\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.5}$$

Для решения прямой задачи кинематики методом ортогональных преобразований была реализована соответствующая программа.

1.2.3. Составление уравнений динамики

Для описания динамики, составим уравнения методом Лагранжа-Эйлера [2, 5]. Обощенными координатами будут являться углы q_1, q_2. Остальные параметры Денавита-Хартенберга выберем следующим образом: $\alpha_1 = \alpha_2 = 0, d_1 = d_2 = 0$. Длины звеньев равны $a_1 = a_2 = l$. Звенья имеют массы m_1, m_2 соответственно. Соответствующая матрица ортогонального преобразования будет иметь вид:

$$H^{0,2} = \begin{bmatrix} C_{12} & -S_{12} & 0 & l(C_{12} + C_1) \\ S_{12} & C_{12} & 0 & l(S_{12} + S_1) \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.6}$$

где $C_i = \cos q_i$ $S_i = \sin q_i$ $C_{ij} = \cos(q_i + q_j)$, $S_{ij} = \sin(q_i + q_j)$

Введем вспомогательную матрицу Q:

$$Q = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{1.7}$$

А также матрицы однородных дифференциальных преобразований U_{ij}:

$$U_{ij} = \frac{\partial H^{0,i}}{\partial Q_j} = QH^{0,j} \tag{1.8}$$

Также расчитаем матрицы псевдоинерций J_i для описания вектора, определяющего влияния вектора силы тяжести:

$$c(q) = \begin{bmatrix} c_2 \\ c_1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2}m_1 glC_1 + \frac{1}{2}m_2 glC_1 2 + m_2 glC_1 \\ \frac{1}{2}m_2 glC_{12} \end{bmatrix} \tag{1.9}$$

Вектор, описывающий центробежное и кориолисово ускорение имеет следующий вид:

$$h(q, \dot{q}) = \begin{bmatrix} -\frac{1}{2}m_2 S_2 l^2 \theta_2^2 - m_2 S_2 l^2 \dot{\theta}\dot{\theta} \\ \frac{1}{2}m_2 S_2 l^2 \dot{\theta}^2 \end{bmatrix} \tag{1.10}$$

Матрица инерции $M(\theta)$ выглядит следующим образом:

$$D(q) = \begin{bmatrix} \frac{1}{2}m_1 l^2 + \frac{4}{3}m_2 l^2 + m_2 C_2 l^2 & \frac{1}{3}m_2 l^2 + \frac{1}{2}m_2 l^2 C_2 \\ \frac{1}{1}m_2 l^2 + \frac{1}{2}m_2 l^2 C_2 & \frac{1}{3}m_2 l^2 \end{bmatrix} \tag{1.11}$$

Таким образом, динамические уравнения 2-х звенного манипулятора будут выглядеть таким образом [2]:

$$\boldsymbol{\tau(t)} = D(q)\ddot{\boldsymbol{q}}(t) + \boldsymbol{h}(q, \dot{q}) + \boldsymbol{c}(q) + \boldsymbol{F}(\dot{q}) \tag{1.12}$$

$F(q)$ - вектор сил трения. В рассматриваемой задаче мы пренебрегаем силами трения.

Здесь τ_i - сила, которую должен развить силовой привод i-го сочленения, чтобы реализовать указанное движение.

Полученная система является нелинейной. При численном интегрировании системы (1.12) [3], можем получить возможное движение при отключенных силовых приводах.

Таким образом, задачу моделирования движения манипулятора можно считать решенной. Полученное уравнение (1.12) можно использовать для синтеза управления, а также для исследования свойств и анализа системы.

1.3. Синтез управления

1.3.1. ПИД-регулятор

Параметры уравнения (1.12) зависят от многих неконтролируемых возмущений, например матрица инерции звеньев при переменной нагрузке и коэффициенты трения

в сочленениях, а следовательно, изменяются во времени с различной интенсивностью. В этой связи разработаны различные подходы адаптации параметров системы управления манипулятором [6].

Рассмотрим непрерывную систему управления на основе классического ПИД-управления [1, 5]. ПИД-регулятор реализует управление:

$$\boldsymbol{\tau}(t) = (\boldsymbol{K}_{p0}) + \Delta\boldsymbol{K}_p(t))\boldsymbol{e}(t) + (\boldsymbol{K_{i0}} + \Delta\boldsymbol{K}_i(t)) \int_0^t \boldsymbol{e}(t)dt + (\boldsymbol{K}_{d0} + \Delta\boldsymbol{K}_d(t))\frac{d\boldsymbol{e}(t)}{dt} \quad (1.13)$$

Здесь $\boldsymbol{K}_{p0}$, $\boldsymbol{K}_{i0}$, $\boldsymbol{K}_{d0}$ - векторы пропорциональных, интегральных и дифференциальных кожффициентов данного регулятора соотвественно. $\Delta\boldsymbol{K}_p(t)$, $\Delta\boldsymbol{K}_i(t)$, $\Delta\boldsymbol{K}_d(t)$ - векторы приращения коэффициентов регулятора. $\boldsymbol{e}(t) = (\boldsymbol{q}_d(t) - \boldsymbol{q}(t))$ вектор ошибки слежения (в радианах). $\boldsymbol{q}_d(t)$ - вектор заданных значений обощенных координат. $\boldsymbol{q}(t)$ - вектор текущих обощенных координат.

Параметры регулятора настраиваются таким образом, чтобы минимизировать ошибку.

$$\boldsymbol{Q}(t) = \frac{1}{2}\boldsymbol{e}^2(t) \to \min_{\Delta\boldsymbol{K}(t)\in\boldsymbol{K}} \quad (1.14)$$

Здесь $\boldsymbol{K} = \Delta\boldsymbol{K}(t) : \Delta\boldsymbol{K}(t) \in E^3, \Delta\boldsymbol{K}_{min} \leqslant \Delta\boldsymbol{K}(t) \leqslant \Delta\boldsymbol{K}_{max}$

Структурная схема представлена на Рис. 1.3.

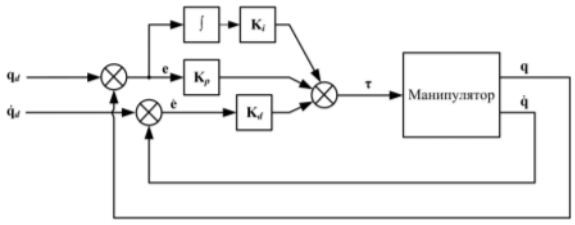

Рис. 1.3. Схема непрерывной системы управления на основе ПИД-регулятора без регулировки

ПИД-регуляторы являются одним из самых распространенных и простых видов регуляторов. Но несмотря на простоту они являются достаточно эффективными и находят широкое применение в реальных системах управления. Однако, задача настройки параметров данного регулятора является весьма нетривиальной. В зависимости от разных параметров заданной траектории — крутость и резкость траектории или наоборот, протяженные прямые отрезки, оптимальными могут быть различные значения

параметров. Далее попробуем провести настройку с помощью нейронной сети. Такой ПИД-регулятор является одним из алгоритмов нейроуправления.

1.4. Нейроупавление

1.4.1. Устройство нейронной сети

Простейшая модель нейронной сети, используемой для настройки параметров ПИД-регулятора - многослойный перцептрон (см. рис. 1.4). Перцептрон состоит из нескольких слоёв нейронов:

1. Входной слой, содержащий псевдо-нейроны, которые передают дальше значения предикторов — параметров объекта;
2. Один или несколько скрытых слоёв;
3. Выходной слой, содержащий один нейрон.

Передача сигналов (активация) нейронной сети происходит от входного слоя, через скрытые слои, к выходному слою.

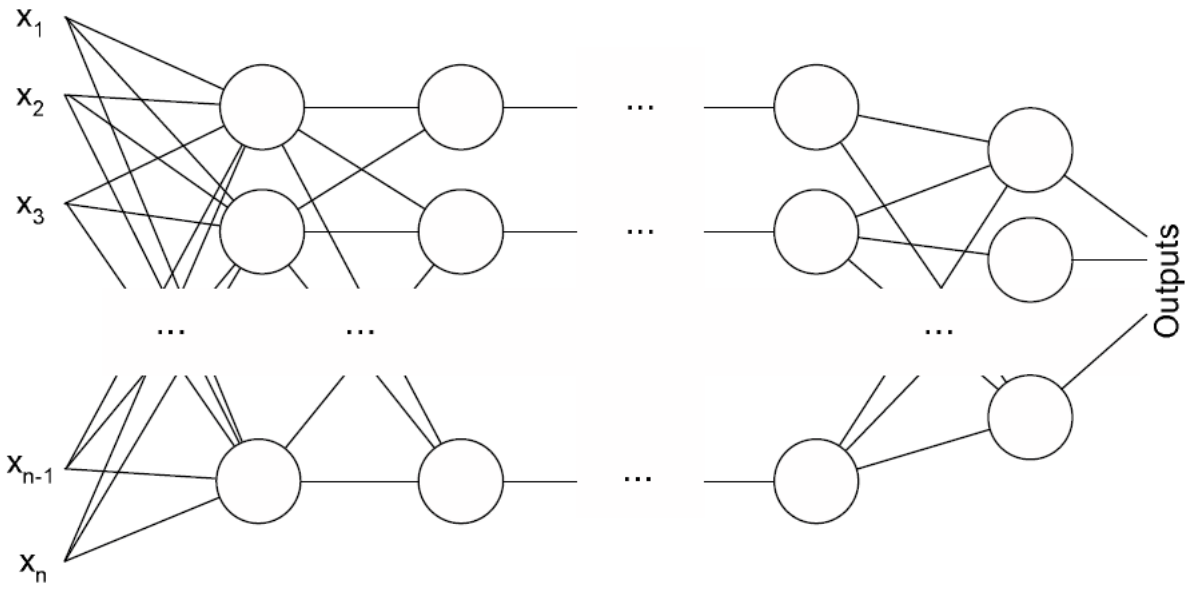

Рис. 1.4. Схема простейшей нейронной сети

Все нейроны (кроме входного слоя) имеют одинаковое строение, состоят из двух частей — сумматорной и активационной функций. Сумматорная функция определяет

то, как нейрон будет использовать входящую информацию из предыдущего слоя. Активационная функция определяет реакцию нейрона, которая будет передана по всем выходным связям в следующий слой.

В качестве сумматорной функции выбрана взвешенная сумма всех входящих сигналов:

$$S = b + \sum_{j=1}^{m} x_j w_j$$

где m — количество входящих сигналов нейрона, x_j — значение, получаемое по j-ому входу, w_j — вес j-ого входа, b — некоторое смещение, изменяемое в процессе обучения. Смещение можно учитывать в сумме, если добавить в каждый слой, кроме выходного на первое место нейрон, у которого значение активации будет всегда равно 1.

В качестве активационной функции выберем логистическую (сигмоидальную) (см. рис. 1.5):

$$\sigma(S) = \frac{1}{1 + e^{-S}}$$

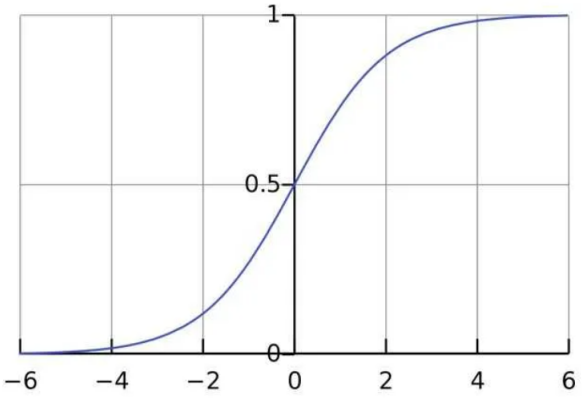

Рис. 1.5. Сигмоидальная функция активации

Логистическая функция является гладкой, что необходимо для работы алгоритма обучения. Кроме того, её значение можно интерпретировать как вероятность принадлежности объекта к одному из двух классов.

Для обучения используется алгоритм обратного распространенния ошибки [7], который основывается на градиентном спуске по простанству весов в сторону уменьшения значений целевой функции ошибки.

Для оценки правдоподобности предсказаний используется квадратичная функция

ошибки:

$$E = \frac{1}{2N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

где N — количество примеров, $\hat{y}_i$ — предсказанное значение для i-ого примера, y_i — правильный ответ для него.

Для того, чтобы понять, как изменится значение функции ошибки при изменении какого-либо веса входящих сигналов нейрона, нужно взять её частную производную по этому весу.

Сначала считается изменение весов в выходном слое:

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j}$$

где $\eta \in \mathbb{R}$ — скорость обучения.

В векторном виде:

$$\Delta W = -\eta \nabla_W E$$

где $\nabla_W E = \left(\frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_m} \right)$ — градиент E в точке W.

Посчитаем частную производную от функции E по j-му весу:

$$\frac{\partial E}{\partial w_j} = \frac{\partial \left(\frac{1}{2N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 \right)}{\partial w_j}$$

Так как производная суммы равна сумме производных, возьмём для простоты один пример, а после просуммируем все значения:

$$\frac{1}{2} \cdot \frac{\partial (\hat{y} - y)^2}{\partial w_j} = \frac{1}{2} \cdot \frac{\partial (\hat{y} - y)^2}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_j} = (\hat{y} - y) \frac{\partial \sigma(S)}{\partial w_j} =$$

$$= (\sigma(S) - y) \sigma'(S) \frac{\partial \sum_{j=1}^{m} x_j w_j}{\partial w_j} = (\sigma(S) - y) \sigma(S) (1 - \sigma(S)) x_j$$

Итак, общая формула для j-ого веса по N примерам:

$$\frac{\partial E}{\partial w_j} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i) \hat{y}_i (1 - \hat{y}_i) x_j$$

Далее полученная ошибка распространяется по ИНС в обратном порядке, от выходного слоя ко входному, изменяя веса скрытых слоёв. Данный алгоритм называется метод обратного распространения ошибки [7].

1.4.2. Настройка параметров ПИД-регулятора с помощью нейронной сети

Подбор параметров ПИД-регялутора будем производить с помощью нейронной сети. При такой настройке параметры вычисляются с помощью однослойной нейронной сети (см. рис. 1.6). Струтурная схема системы управления с настройкой параметров предствалена ниже (см. рис. 1.7).

В качестве алгоритма оптимизации весов будем использовать метод обратного распространения ошибки на основе градиентного спуска [6, 7].

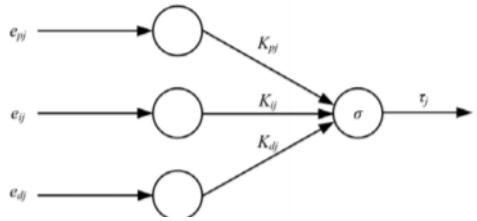

Рис. 1.6. Структура нейронной сети ПИД-регулятора

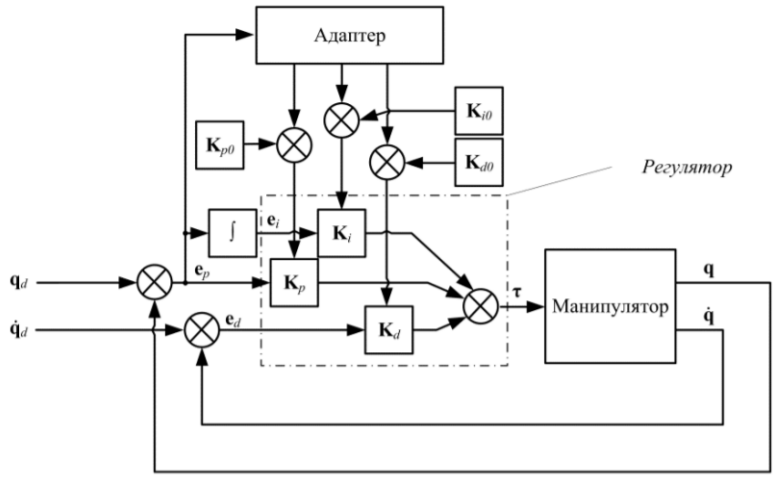

Рис. 1.7. Структурная схема адаптивной системы управления с настройкой параметров ПИД-регулятор с помощью нейронной сети

Выражения для настройки выглядят следующим образом:

$$K_{pj} = K_{pj_0} + \eta_{pj} \int\limits_0^t \frac{\partial Q_j}{\partial K_{pj}} dt, \tag{1.15}$$

$$K_{ij} = K_{ij_0} + \eta_{ij} \int\limits_0^t \frac{\partial Q_j}{\partial K_{ij}} dt, \tag{1.16}$$

$$K_{dj} = K_{dj_0} + \eta_{dj} \int\limits_0^t \frac{\partial Q_j}{\partial K_{dj}} dt \tag{1.17}$$

Здесь η_{pj}, η_{ij}, η_{dj} - векторы скорстей обучений для пропорциональной, интегральной и дифференциальной составляющх для j-х сочленений.

Алгоритм на основе нейронной сети является нелинейным и способен обучаться в ходе работы.

1.5. Сравнение результатов

Сравним результаты для двухзвенного манипулятора со следующими параметрами: $a_1 = 1$, $a_2 = 1$, $m_1 = 2$, $m_2 = 0.5$, $m_l = 3.75$, $B = 0.3$. a_j - длина j-го звена, m_j - его масса, m_l - масса груза, B - коэффициент трения.

Заданная траектория имеет вид:

$$q_{d1} = 20sin(\omega t), q_{d2} = 15sin(\omega t), \omega = 0.2\pi \tag{1.18}$$

Эффективность будем измерять для каждого звена следущим образом:

$$J_i = \frac{1}{N} \sum_{k=0}^{N} \sqrt{e_j^2(k)}, j = 1, 2 \tag{1.19}$$

1.5.1. Классический ПИД-регулятор без адаптации

Значения коэффициентов равны $\boldsymbol{K}_{p0} = (1000, 1000)$, $\boldsymbol{K}_{i0} = (100, 100)$, $\boldsymbol{K}_{d0} = (75, 75)$, $\Delta\boldsymbol{K}_p = \Delta\boldsymbol{K}_i = \Delta\boldsymbol{K}_d = (0, 0)$.

На графиках (см. рис. 1.8) мы видим, что ошибка первого звена всюду больше ошибки второго звена.

Эффективность $\boldsymbol{J} = (J_1, J_2) = (2.1635, 0.9102)$

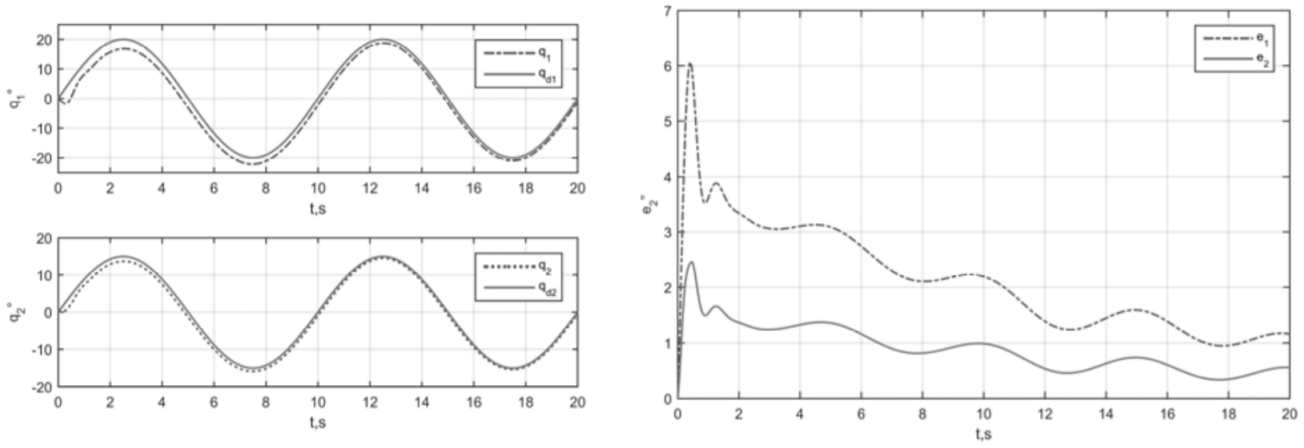

Рис. 1.8. Качество слежения: а - заданные и реальные обощенные координаты, б - ошибки слежения в звеньях

1.5.2. Настройка параметров ПИД с помощью нейронной сети

Нейронная сеть (рис. 2.2) оптимизирует парметры ПИД-регулятора. Начальные значения коэффициентов: $\boldsymbol{K}_{p0} = (1000, 1000)$, $\boldsymbol{K}_{i0} = (100, 100)$, $\boldsymbol{K}_{d0} = (75, 75)$. Векторы скоростей обучения: $\boldsymbol{\eta}_p = (600, 800)$, $\boldsymbol{\eta}_i = (500, 600)$, $\boldsymbol{\eta}_d = (800, 800)$

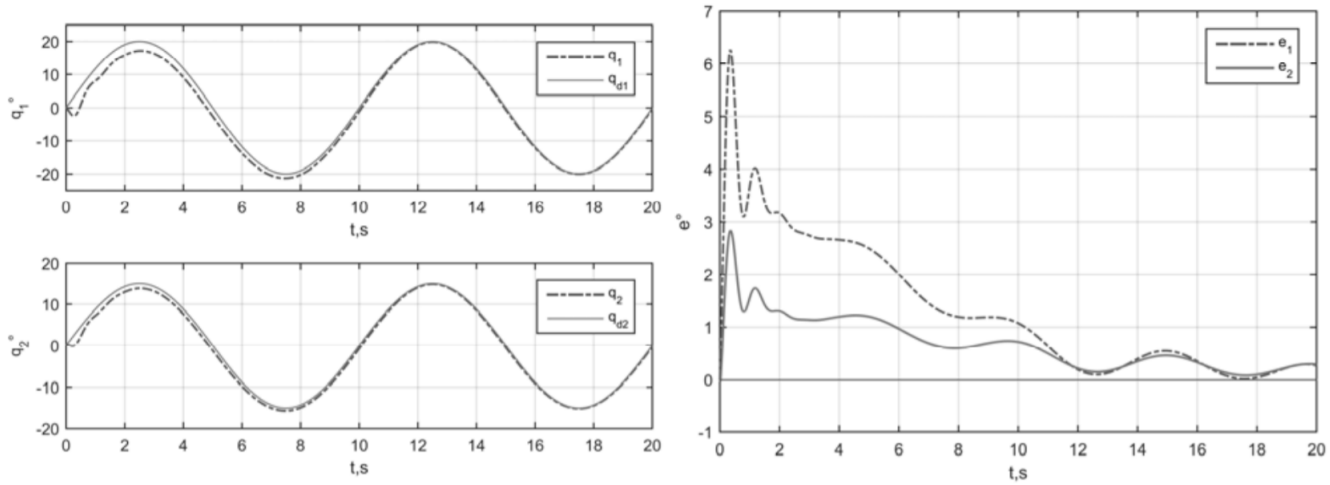

Рис. 1.9. Качество слежения: а - заданные и реальные обощенные координаты, б - ошибки слежения в звеньях

Как мы можем увидеть на графиках (см. рис. 1.9), ошибка находится примерно на том же уровне, что и в случае обычного ПИД-регулятора, но доходит до минимальных значений в 2 раза быстрее

Эффективность $\boldsymbol{J} = (J_1, J_2) = (1, 4089, 0.7173)$

1.6. Вывод

Было проведено меделирование плоского манипулятора. Уравнения динамики были полученными методом Лагранжа-Эйлера. Данный метод удобнее использовать для составления уравнений и дальнейшего анализа систем с небольшим числом степеней свободы.

Были применены методы нейроуправления. ПИД-регулятор, настраиваемый с помощью нейросети не дает ощутимого преимущества в точности, но позволяет лучше адаптироваться к новым траекториям.

Глава 2

Моделирование движения 7-ми звенного манипулятора

2.1. Постановка задачи

Далее будем рассматривать более сложный 7-звенный мманипулятор kuku iiwa (см. рис. 2.1). Геометрия записывается в форме параметров Денавита-Хартенберга. Динамические параметры, такие как моменты инерции звеньев и их массы считаются известными заранее.

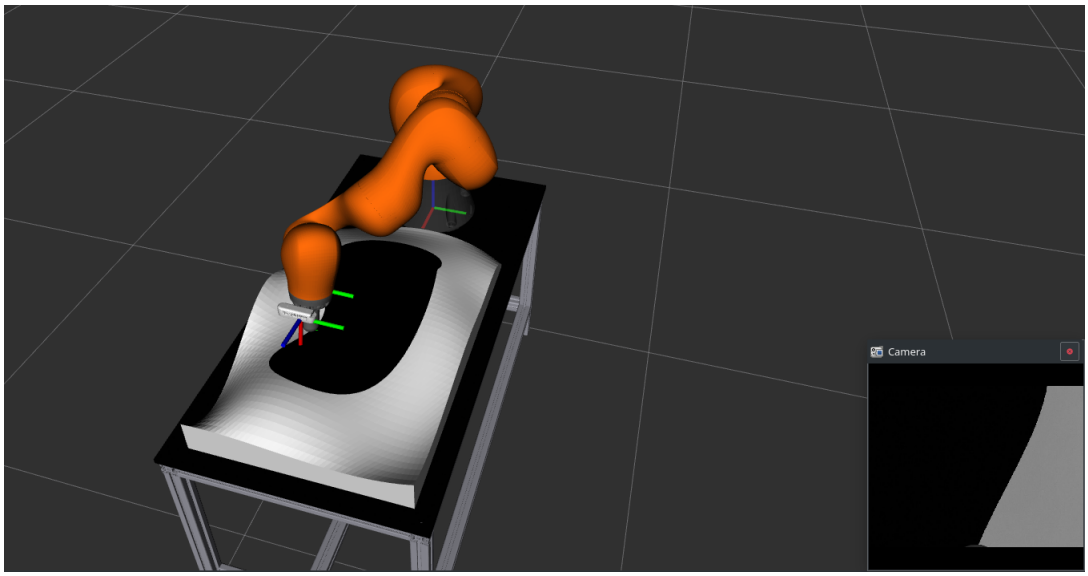

Рис. 2.1. Визуализация манипулятора kuku iiwa

В случае данного манипулятора задача усложняется. Представим, что нам необходимо соединить криволинейную поверхность с помощью сварки. Рабочий орган в данной ситуации должен давить с определенной силой, которую мы можем задать заранее. Траекторию заранее мы не знаем, а можем получить её только с помощью изображений с камеры.

2.2. Геометрия манипулятора

Кинематическая схема манипулятора iiwa представлена на рис. 2.2. Оси кинематических пар выбраны методом, описанным в главе 1.

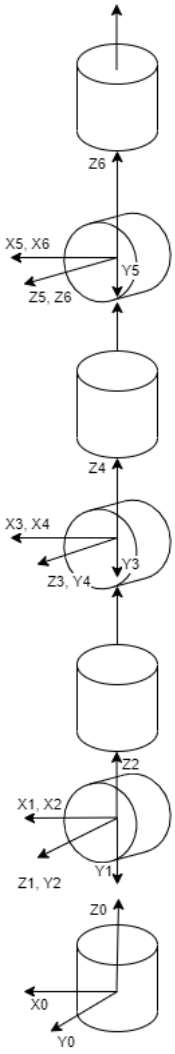

Рис. 2.2. Кинематическая схема манипуялтора iiwa

Параметры Денавита-Хартенберга будут выглядеть следующим образом:

	θ_i	d_i	a_i	α_i
1	q_1	d_1	0	-$\pi/2$
2	q_2	0	0	$\pi/2$
3	q_3	d_3	0	-$\pi/2$
4	q_4	0	0	$\pi/2$
5	q_5	d_5	0	-$\pi/2$
6	q_6	0	0	$\pi/2$
7	q_7	d_7	0	0

Таблица 2.1. Параметры Денавита-Хартенберга для 7-звенного манипулятора

2.3. Уравнения динамики

2.3.1. Рекурсивные уравнения Ньютона-Эйлера

В данном разделе выводятся уравнения для 7-звенного пространственного манипулятора. Напомим общий вид уравнения движения в конфигурационном пространстве в векторном виде:

$$\boldsymbol{\tau(t)} = D(q)\ddot{\boldsymbol{q}}(t) + \boldsymbol{h}(q,\dot{q}) + \boldsymbol{c}(q) + \boldsymbol{F}(\dot{q}) \tag{2.1}$$

Динамическую модель получим с помощью рекурсивных уравнений Ньютона-Эйлера [2, 5].

Необходимые данные для процедуры составления уравнений:

- Количество звеньев и их тип. В случае рассматриваемого манипулятора - все звенья вращательные
- Кинематические параметры (параметры Денавита-Хартенберга)
- Динамические параметры: массы звеньев, координаты центров масс, инерции.

Построение динамической модели методом Ньютона-Эйлера разбивается на несколько этапов. Первый этап — прямая рекурсия (см. рис 2.3) (от базы к последнему звену). Вычисляем ω_i, $\dot{\omega}_i$ - вращательные скорости и ускорения. Ускорения сочленений a_i, а также ускорения центров масс звеньев a_{ci}.

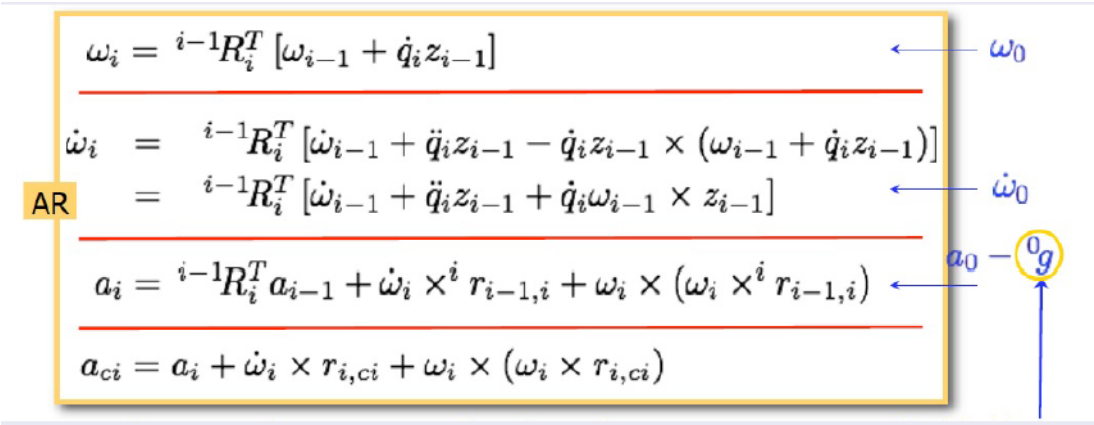

Рис. 2.3. Прямая рекурсия метода Ньютона-Эйлера

Здесь ${}^{i-1}R_i^T$ — матрица перехода от (i-1) системы координат к i-ой. z_i — единичные векторы осей шарниров. Задаются в (i-1)-й системе координат. $r_{i-1,i}$ — вектор i-го

сочленения в системе координат (i-1)-го . $r_{i,ci}$ — вектор центра масс звена в i-й системе координат.

Следующий этап - обратная рекурсия (от последнего звена к базе). На каждой итерации мы узнаем соответствующие моменты τ_i и силы f_i, действующие на i-е звено (см. рис. 2.4). Здесь в выражениях учитываются массы m_i и моменты инерции I_i i-го звена

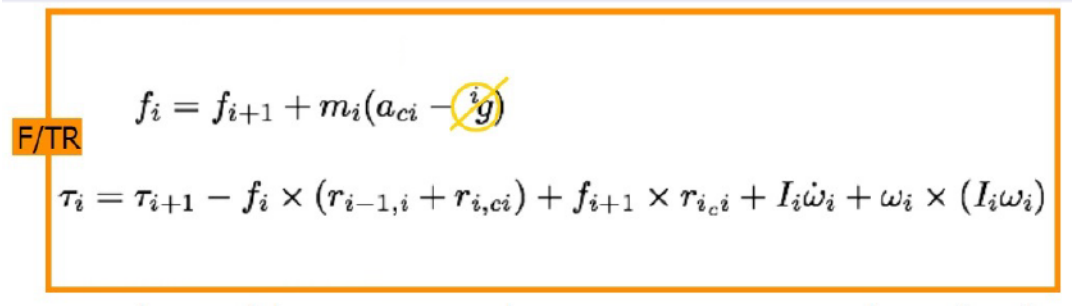

Рис. 2.4. Обратная рекурсия метода Ньютона-Эйлера

Третий этап процедуры — проекция моментов. С помощью данной операции мы можем уточнить управляющее воздействие.

$$u_i = \begin{cases} f_i^T \cdot z_{i-1} & \text{Для линейных звеньев} \\ \tau_i^T \cdot z_{i-1} & \text{Для вращательных звеньев} \end{cases} \tag{2.2}$$

Общая схема процедуры изображена на рис. 2.5:

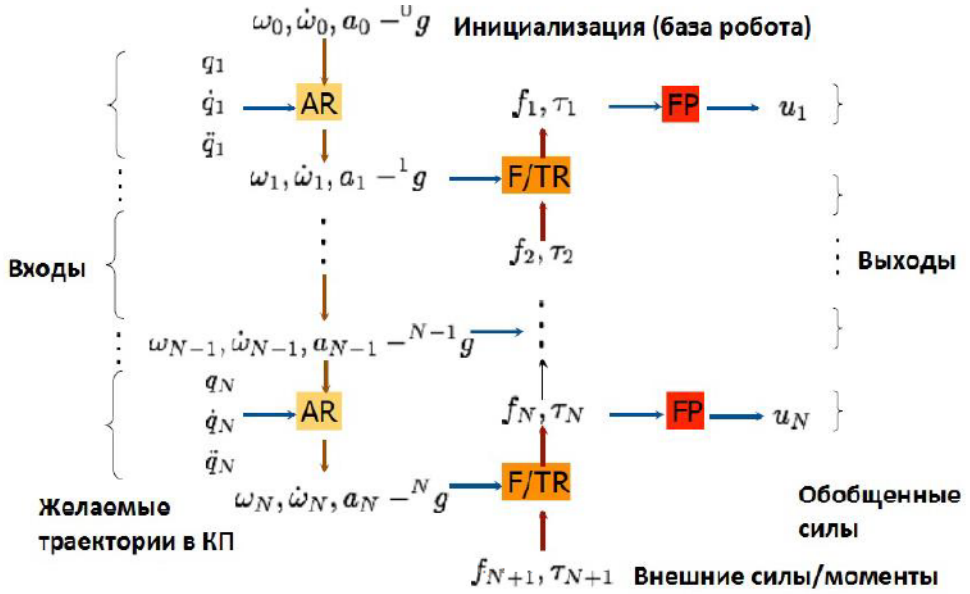

Рис. 2.5. Обратная рекурсия метода Ньютона-Эйлера

Для вывода уравнений динамики с помощью алгоритма Ньютона-Эйлера была составлена программа в Wolfram Mathematica appCB.
С помощью различных подстановок данных $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$ в алгоритм Ньютона-Эйлера, мы можем получить матрицу $\boldsymbol{D}$, $\boldsymbol{h}$ и вектор $\boldsymbol{c}$:

- Полная динамика $\tau = NE(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$
- Вектор гравитации $c(\boldsymbol{q}) = NE(\boldsymbol{q}, 0, 0)$
- Корилисовы и центробежные силы $h(\boldsymbol{q}, \dot{\boldsymbol{q}}) = NE(\boldsymbol{q}, \dot{\boldsymbol{q}}, 0)|_{g=0}$
- Матрица инерции (постолбцово) $d_i(\boldsymbol{q}) = NE(\boldsymbol{q}, 0, 1)|_{g=0}$
- Обобщенный момент $M(\boldsymbol{q})\dot{\boldsymbol{q}} = NE(\boldsymbol{q}, 0, \dot{\boldsymbol{q}})|_{g=0}$

В результате работы алгоритма Ньютона-Эйлера мы получили уравнения динамики, с помощью которых можно реализовать управление движением манипулятора.

2.4. Планирование траектории

Для решения задачи планирования траектории одним из самых распространенных подходов является метод интерполяции с помощью сплайн-функций [3]. Суть данного метода заключается в расчете промежуточных значений обощенных координант, скоростей и ускорений для каждого звена между каждыми конфигурациями с помощью полиномов вида [1]:

$$q_i(t) = a_{l,i}t^l + al - 1, it^{l-1} + \cdots + a_{2,i}t^2 + a_{1,i}t + a_{0,i}, \tag{2.3}$$

$$\dot{q}_i(t) = la_{l,i}t^{l-1} + (l-1)al - 1, it^{l-2} + \cdots + 2a_{2,i}t + a_{1,i}, \tag{2.4}$$

$$\ddot{q}_i(t) = l(l-1)a_{l,i}t^{l-2} + (l-1)(l-2)al - 1, it^{l-3} + \cdots + 2a_{2,i}, \tag{2.5}$$

Исходными данными в задаче планирования траектории является набор конфигураций и моменты их прохождения. Набор конфигураций нам заранее неизвестен, мы получаем его в виде последовательности точек по изображению с камеры на конце манипулятора с помощью фильтрации изображения [1]. Далее переносим координаты точек из системы координат камеры в глобальную систему координат [8].

Общий алгоритм построения траекторий с помощью сплайн функций:

- Определяем число сегментов. В нашем случае число сегментов вычислить заранее нельзя. Аппроксимируем последовательно каждые 3 полученные с камеры точки трехсегментной траекторией.
- Нормируем время.
- Определяем граничные условия и требования на непрерывность между сегментами
- Составляем матричное уравнение и разрешаем его относительно неизвестных коэффициентов полиномов.

Стоит отметить, что полиномы (2.3-2.5) строятся для каждой обобщенной координаты. Более подробно процесс описан в соответствующих параграфах в книгах [1] и [8].

2.5. Управление по силе

Для решения задачи управления рассматриваемым манипулятором был выбран алгоритм непрямого управления по силе. В данных уравнениях мы можем регулировать коэффициенты демпфирования и жесткости, K_D и K_P соответственно, для получения оптимального и устойчивого управления.

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q}) + G(q) = u - J(q)^T F, \tag{2.6}$$

$$u = M(q)a_q + C(q,\dot{q})\dot{q} + G(q) + J(q)^T \tilde{F}, \tag{2.7}$$

$$a_q = J_a^- 1(q)(a_X - \dot{J}_a(q,\dot{q})\dot{q}), \tag{2.8}$$

$$a_X = \ddot{X}_d(t) + M_d^- 1(K_D\dot{\tilde{X}} + K_P\tilde{X} - F_a) \tag{2.9}$$

Здесь левая часть уравнения (2.6) — уравнение динамики в матричном виде в пространстве состояния, полученное методом Ньютона-Эйлера в предыдущих пунктах. u — управляющее воздействие, $J(q)$ — аналитический якобиан [1]. F — внешняя сила, возникающая из-за контакта с поверхностью. $\tilde{F}$ — оценка внешний силы или измерение, которое мы можем получить с датчика. $\ddot{X}_d$ - желаемое ускорение в операционном пространстве, полученное на этапе планирования траектории. $\tilde{X} = X_d - X$, $\dot{\tilde{X}} = \dot{X}_d - \dot{X}$ — отклонения желаемой траектории и скорости от текущих. F_a — желаемая сила контакта с поверхностью.

Основная суть данного управления — вычисление отклонения желаемого ускорения a_q в конфигурационном пространстве. Разность $a_X - \dot{J}_a(q, \dot{q})\dot{q}$ отражает данное отклонение в операционном пространстве.

Важным моментом является то, что полное ускорение в операционном пространстве вычисляется как:

$$\ddot{X} = \dot{J}(q, \dot{q})\dot{q} + J(q)\ddot{q} \tag{2.10}$$

Но вторым слагаемым мы можем пренебречь ввиду его малости.

Отдельно стоит отметить, что уравнения динамики вычислены с помощью вспомогательных библиотек символьно. Но существует и другой подход. Все символьные вычисление в алгоритме Ньютона-Эйлера можно заменить численными и на каждой итерации уточнять текущую траекторию численно. Для некоторых алгоритмов управления такой подход может быть менее затратным в плане вычислительных ресурсов.

2.6. Программная реализация и визуализация

2.6.1. Описание основных элементов программы

Основные файлы программной реализации можно увидеть в приложении А и Б.

Файл *camera.py* отвечает за обработку и фильтрацию изображения, полученного с помощью камеры манипулятора. Переводятся координаты точек из системы координат камеры в глобальную систему координат.

В файле *iiwa_control.py* производится считывание и преобразование данных о манипуляторе, решение прямой задачи кинематики, составление уравнений динамики на основе геометрических и динамических характеристик манипулятора, а также реализация предложенного имеданского регулятора.

2.6.2. Использованные инструменты

Для визуализации и составления кинематической схемы манипулятора модели использовалась среда Rviz. Также в данной среде создана сцена с рабочим столом и рабочей поверхностью.

С помощью фреймворка Ros и среды Gazebo были описаны основные структурные элементы, такие как звенья манипулятора, сочленения, камера манипулятора, а также их поведение и взаимодействие [9, 10].

Уравнения динамики и их решения получены с помощью библиотеки pyKDL и заранее прописанных динамических и кинематических параметров манипулятора.

Обработка изображения с камеры осуществлялась с помощью библиотеки CV2.

2.7. Вывод

Построение модели в различных средах для визуализации робототехнических систем позволяет выполнить моделирование и проверку системы управления на работоспособность без создания реального прототипа. Подобное моделирование и визуализация позволяет уточнять динамические характеристики модулей, а также исправлять недостатки манипулятора на стадии проектирования.

С помощью представленой модели и составленной программы можно тестировать различные алгоритмы управления, а также уточнять параметры данных алгоритмов без необхходимость загружать программы в реальный контроллер, что значительно ускоряет разработку оптимальных систем управления.

Заключение

В данной работе было проведено численное моделирование 2-х звенного плоского и пространственного 7-ми звенного робота манипулятора. Использовались 2 основных метода составления уравнений динамики: Лагранжа-Эйлера и Ньютона-Эйлера.

Динамические уравнения для плоского данного манипулятора были получены методом Лагранжа-Эйлера. Для данного манипулятора был построен классический ПИД-регулятор, а также ПИД-регулятор, настраиваемый с помощью нейронной сети - использовался косвенный подход применения нейронной сети.

Исследуемое применение нейронных сетей не дает большего выигрыша в точности, но, исходя из полученных результатов, можно сделать вывод о том, что применение нейронных сетей положительно влияет на скорость уменьшения ошибки и адаптацию параметров регулятора к новым траекториям.

Динамические уравнения для 7-ми звенного манипулятора были выведены с помощью метода Ньютона-Эйлера, так как данный метод лучше подходит для программной реализации.

Для 7-ми звенного была построена визуализация в среде Gazebo. На языке Python с использованием фреймворка Ros и различных библиотек (openCV, pyKdl и др.) была построена модель манипулятора и реализован алгоритм непрямого управления по силе. Манипулятор справляется с поставленной задачей. Код программ и соответствующее окружение загружены на GitHub

Также, была написана программа в пакете Wolfram Mathematica appCB, реализующая алгоритм Ньютона-Эйлера для произвольного n-звенного манипулятора, а также вычисляющая компоненты матриц в уравнении динамики.

Приложение А: Пример кода обработки изображения

```
#!/usr/bin/env python3
import rospy
from cv_bridge import CvBridge, CvBridgeError
from sensor_msgs.msg import Image
from geometry_msgs.msg import Point
import cv2
import numpy as np
from matplotlib import pyplot as plt
class image_processor:
    def __init__(self):
        self.bridge = CvBridge()
        self.image_sub = rospy.Subscriber("/iiwa/camera1/image_raw", \
        Image, self.callback)
        self.point_pub = rospy.Publisher("/iiwa/camera1/line_coordinate",\
        Point, queue_size=2)
        self.x0 = 480
        self.y0 = 320
        self.point = Point(0, 0, 0)
        self.vel = 1
    def callback(self, data):
        try:
            cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
        except CvBridgeError as e:
            print(e)
        gray = cv2.cvtColor(cv_image[:-100, :-200], cv2.COLOR_BGR2GRAY)
        edges = cv2.Canny(gray, 100, 200)
        white_poisnts = np.argwhere(edges == 255)
        p = (0, 0)
        x = [0, 0]
        norm = 1.0
        if white_poisnts.size > 0:
```

```
            p = (white_poisnts[-1, 1], white_poisnts[-1, 0])
            x = [p[1] - self.x0, -p[0] + self.y0]
            norm = np.sqrt(x[0]**2 + x[1]**2)
        self.point.x = self.vel*x[0]/norm
        self.point.y = self.vel*x[1]/norm
        self.point_pub.publish(self.point)
        cv_image = cv2.circle(cv_image, p, radius = 5, color=(0, 0, 255),\
        thickness=-1)
        cv2.imshow("Image window", cv_image)

if __name__ == '__main__':
    rospy.init_node('iiwa_camera_processor')
    processor = image_processor()
    rospy.loginfo("HI")

    try:
        rospy.spin()
    except KeyboardInterrupt:
        print("Shutting down")
```

Приложение B: Пример кода реализации управления движения

```
#!/usr/bin/env python2
# Common packages
import numpy as np
# Major ROS
import rospy
# Orocos Kinematic Dynamic Library (KDL)
import PyKDL
import kdl_parser_py.urdf as urdf
# Services
from controller_manager_msgs.srv import SwitchController, SwitchControllerRequest
# Messages
from std_msgs.msg import Float64
from sensor_msgs.msg import JointState

class iiwa:
def __init__(self):
self.base_link = 'iiwa_link_0'
self.end_effector = 'tool_link_ee_kuka'
flag, self.tree = urdf.treeFromParam('/robot_description')
self.arm_chain = self.tree.getChain(self.base_link, self.end_effector)
# Gravity vector
self.gravity = PyKDL.Vector(0, 0, -9.80) # From gazebo world
# The names of joints
self.joint_names = np.array(['iiwa_joint_{}'.format(i) for i in range(1, 8)])
# Number of joints (7)
self.n = self.arm_chain.getNrOfJoints()
# Joint positions and velocities
self.joint_positions_kdl = PyKDL.JntArray(self.n)
self.joint_positions = np.zeros([self.n, 1])
self.joint_velocities_kdl = PyKDL.JntArray(self.n)
self.joint_velocities = np.zeros([self.n, 1])
```

```
# Kinematics and dynamics of the robot
# Methods allow to get the Jacobian matrix
# and Dynamic model matrices as M (Inertia matrix)
#C (Corioliss matrix) and G (Gravity matrix)
self.jacobian_kdl = PyKDL.ChainJntToJacSolver(self.arm_chain)
self.dynamics_kdl = PyKDL.ChainDynParam(self.arm_chain, self.gravity)
# Dynamic model matrices
self.G = PyKDL.JntArray(self.n)
self.Cdq = PyKDL.JntArray(self.n)
self.M_kdl = PyKDL.JntSpaceInertiaMatrix(self.n)
self.M = np.mat(np.zeros((self.n, self.n)))


# Publishers and Subscribers
self.joint_states_sub = rospy.Subscriber("/iiwa/joint_states",\
JointState, self.update_joints_state)
self.torque_control_pub = \
[rospy.Publisher('/iiwa/joint{}_torque_controller/command'.format\
(i), Float64, queue_size = 10) for i in range(1, self.n+1)]
# Test angles
self.pos_test = np.zeros([self.n, 1])
self.pos_test[0] = 0
self.pos_test[1] = 0.5
self.pos_test[2] = 0
self.pos_test[3] = -0.5
self.pos_test[4] = 0
self.pos_test[5] = 2
self.pos_test[6] = 0
self.err = self.pos_test
# Control torque
self.tau = np.ones([self.n, 1])
self.Kp = np.identity(self.n)*20
```

```
self.Kd = np.identity(self.n)*1
# self.torque_control_pub[0].publish(0.0)
# tmp = self.Kp.dot(self.joint_positions)
def update_joints_state(self, joint_state_msg):
for i in range(self.n):
self.joint_positions_kdl[i] = joint_state_msg.position[i]
self.joint_positions[i] = joint_state_msg.position[i]
self.joint_velocities_kdl[i] = joint_state_msg.velocity[i]
self.joint_velocities[i] = joint_state_msg.velocity[i]
# Control
self.dynamics_kdl.JntToGravity(self.joint_positions_kdl, self.G)
self.dynamics_kdl.JntToCoriolis(self.joint_positions_kdl, \
self.joint_velocities_kdl, self.Cdq)
self.dynamics_kdl.JntToMass(self.joint_positions_kdl, self.M_kdl)
self.kdl_to_mat(self.M_kdl, self.M)
self.tau = (self.Kp.dot(self.pos_test - self.joint_positions))
for i in range(self.n):
self.torque_control_pub[i].publish(self.tau[i] - self.Kd[i, i]*\
self.joint_velocities_kdl[i] + self.Cdq[i] + self.G[i])
def kdl_to_mat(self, mat_in, mat_out):
for i in range(mat_in.rows()):
for j in range(mat_in.columns()):
mat_out[i,j] = mat_in[i,j]
def setupControlManager():
# Turn on torque control
rospy.loginfo('Turn on torque controllers')
# Create Client object
switchTorqueController = rospy.ServiceProxy('/iiwa/controller_manager/switch_contro
SwitchController)
# Fill the fields
start_controllers =  \
np.array(['joint{}_torque_controller'.format(i) for i in range(1, 8)])
```

```
stop_controllers = np.array([])
strictness = SwitchControllerRequest.BEST_EFFORT
start_asap = False
timeout = 0 # Infinite
# Send service message
resp = switchTorqueController(start_controllers, stop_controllers, \
strictness, start_asap, timeout)
# Print out the response information
rospy.loginfo(resp)

if __name__ == '__main__':
rospy.init_node('iiwa_force_control')
setupControlManager()
rospy.sleep(1.0)
iiwa_robot = iiwa()
rospy.spin()
```

Приложение C: Составление уравнений динамики.

$ee_1:=\{0,1,0\}; ee_2:=\{0,-1,0\};$

$ee_3:=\{0,1,0\}; ee_4:=\{0,-1,0\};$

$ee_5:=\{0,1,0\}; ee_6:=\{0,-1,0\};$

$ee_7:=\{0,0,1\}; rc_0:=\{0,0,0\};$

$rc_1:=\{0,0,d_1/2\}; rc_2:=\{0,-d_3/3,0\};$

$rc_3:=\{0,0,2*d_3/3\};$

$rc_4:=\{0,-d_5/3,0\}; rc_5:=\{0,0,2*d_5/3\};$

$rc_6:=\{0,-d_7/3,0\}; rc_7:=\{0,0,d_7/2\};$

$r_0:=\{0,0,0\}; r_1:=\{0,0,d_1\};$

$r_2:=\{0,-d_3/2,0\}; r_3:=\{0,0,d_3\};$

$r_4:=\{0,-d_5/2,0\}; r_5:=\{0,0,d_5\};$

$r_6:=\{0,-d_7/3,0\};$

$r_7:=\{0,0,d_7\};$

, :

$q:=\{\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6,\theta_7\};$

$q':=\{\theta_1',\theta_2',\theta_3',\theta_4',\theta_5',\theta_6',\theta_7'\};$

$q":=\{\theta_1",\theta_2",\theta_3",\theta_4",\theta_5",\theta_6",\theta_7"\};$

-

$\alpha:=\left\{-\frac{\pi}{2},\frac{\pi}{2},-\frac{\pi}{2},\frac{\pi}{2},-\frac{\pi}{2},\frac{\pi}{2},0\right\};$

i- (i-1)- (Ai-1, i):

A1[i_]:=

$$\begin{pmatrix} \text{Cos}[q[[i]]] & -\text{Cos}[\alpha[[i]]]\text{Sin}[q[[i]]] & \text{Sin}[\alpha[[i]]]\text{Sin}[q[[i]]] \\ \text{Sin}[q[[i]]] & \text{Cos}[\alpha[[i]]]\text{Cos}[q[[i]]] & -\text{Sin}[\alpha[[i]]]\text{Cos}[q[[i]]] \\ 0 & \text{Sin}[\alpha[[i]]] & \text{Cos}[\alpha[[i]]] \end{pmatrix}$$

-

A2[i_]:=Transpose[A1[i]]

$\omega[0]:=\{0,0,0\};$

$\omega[j_]:=A2[j].(\omega[j-1]+q'[[j]]ee_j)$

MatrixForm[$\omega[1]$] MatrixForm[$\omega[2]$] MatrixForm[$\omega[3]$]

$\epsilon[0]:=\{0,0,0\};$

$\epsilon[\mathrm{j\_}]:=$

$\mathrm{A2}[j].\,(\epsilon[j-1]+q"[[j]]\mathrm{ee}_j+q'[[j]]\omega[j-1]\times \mathrm{ee}_j)$

$\mathrm{MatrixForm}[\epsilon[1]]$

:

$\mathrm{Wo}[0]:=\{0,0,0\};$

$\mathrm{Wo}[\mathrm{j\_}]:=$

$\mathrm{A2}[j].\,(\mathrm{Wo}[j-1]+\epsilon[j-1]\times r_{j-1}+\omega[j-1]\times(\omega[j-1]\times r_{j-1}))$

$\mathrm{For}[i=1,i<8,i{+}{+},\mathrm{Print}[\text{“Wo”},i,\text{“ = ”},\mathrm{Simplify}[\mathrm{Wo}[i]]]]$

$\mathrm{Wc}[\mathrm{j\_}]:=\mathrm{Wo}[j]+\epsilon[j]\times \mathrm{rc}_j+\omega[j]\times(\omega[j]\times \mathrm{rc}_j)\,;$

$\mathrm{For}[i=1,i<8,i{+}{+},$

$\mathrm{Print}[\text{“Wc”},i,\text{“ = ”},\mathrm{Simplify}[\mathrm{Wc}[i]]]]$

-.

,

$f_8:=\{0,0,0\};\mathrm{mom}_8:=\{0,0,0\};$

$\mathrm{A1}[8]:=\mathrm{IdentityMatrix}[3]$

, ,

$M:=\{m_1,m_2,m_3,m_4,m_5,m_6,m_7\}\,;G:=\{0,0,g\}$

$\mathrm{Ic}_{\mathrm{i\_}}:=\begin{pmatrix}\mathrm{Jx}_i & 0 & 0\\ 0 & \mathrm{Jy}_i & 0\\ 0 & 0 & \mathrm{Jz}_i\end{pmatrix};$

.

i- Aoi

$\mathrm{Aoi}[1]:=\mathrm{A1}[1];\ \mathrm{Aoi}[\mathrm{i\_}]:=\mathrm{Aoi}[i-1].\mathrm{A1}[i]$

$\mathrm{Aio}[\mathrm{i\_}]:=\mathrm{Transpose}[\mathrm{Aoi}[i]]$

( )

$f_{\mathrm{i\_}}:=\mathrm{ExpandAll}\,[f_{i+1}-m_iG+m_i\mathrm{Aoi}[i].\mathrm{Wc}[i]]$

$\mathrm{For}[k=8,k>0,k{-}{-},$

$\mathrm{Print}[\mathrm{f},k,\text{“ = ”},$

$\mathrm{Collect}\,[f_k,\{\theta_1",\theta_2",\theta_3",\theta_4",\theta_5",\theta_6",$

$\theta_7",(\theta'_1)^2,(\theta'_2)^2,(\theta'_3)^2,(\theta'_4)^2,(\theta'_5)^2,$

$(\theta'_6)^2,(\theta'_7)^2,\theta'_1\theta'_2,\theta'_1\theta'_3,\theta'_1\theta'_4,\theta'_1\theta'_5,$

```
θ'1θ'6, θ'1θ'7, θ'2θ'3, θ'2θ'4, θ'2θ'5, θ'2θ'6,
θ'2θ'7, θ'3θ'4, θ'3θ'5, θ'3θ'6, θ'3θ'7, θ'4θ'5,
 θ'4θ'6, θ'4θ'7, θ'5θ'6, θ'5θ'7, θ'6θ'7} , Simplify]]]
        ( )
mom_i_ :=
ExpandAll [A1[i + 1].mom_{i+1} + r_i × Aio[i].f_{i+1} + rc_i × m_i Wc[i] −
 Aio[i]. (m_i G) + Ic_i.ϵ[i] + ω[i] × (Ic_i.ω[i])]
For[k = 8, k > 0, k−,
Print["mom", k, " = ",
Collect [f_k, {θ1", θ2", θ3", θ4", θ5", θ6",
θ7", (θ'1)^2, (θ'2)^2, (θ'3)^2, (θ'4)^2, (θ'5)^2,
(θ'6)^2, (θ'7)^2, θ'1θ'2, θ'1θ'3, θ'1θ'4, θ'1θ'5,
θ'1θ'6, θ'1θ'7, θ'2θ'3, θ'2θ'4, θ'2θ'5, θ'2θ'6,
θ'2θ'7, θ'3θ'4, θ'3θ'5, θ'3θ'6, θ'3θ'7, θ'4θ'5,
 θ'4θ'6, θ'4θ'7, θ'5θ'6, θ'5θ'7, θ'6θ'7} , Simplify]]]
```

Список литературы

1. О. И. Борисов В. С. Громов А. А. Пыркин. Методы управления робототехническими приложениями. — 2016. — Режим доступа: https://books.ifmo.ru/file/pdf/2094.pdf.
2. Колюбин С. А. Динамика робототехнических систем. — 2017. — С. 36–49. — Режим доступа: https://books.ifmo.ru/file/pdf/2267.pdf.
3. Калиткин Н. А. Численные методы. — 2011.
4. Зубов В. И. Лекции по теории управления. — 1975. — Режим доступа: https://ia803105.us.archive.org/0/items/B-001-026-834-PDF-035/pdf035.pdf.
5. Фу К. Гонсалес Р. Ли К. Робототехника. — 1989.
6. Razmi H. Kashtiban A.M. Nonlinear PID-based analog neural network control for a two link rigid robot manipulator and determining the maximum load carrying capacity // International Journal of Soft Computing and Engineering. — 2012. — Access mode: https://books.ifmo.ru/file/pdf/2094.pdf.
7. Rumelhart D.E. Hinton G.E. Williams R.J. Learning Internal Representations by Error Propagation. // Parallel Distributed Processing, vol. 1. — 1986.
8. Corke Peter. Robotics, vision and control: fundamental algorithms in MATLAB // Springer. — 2017.
9. Lynch Kevin M, Park Frank C. Modern Robotics. // Cambridge University Press. — 2017.
10. Spong. Mark W. Robot modeling and control. — 2006.